Requirements

- Functional: describe the interactions between the system and its environment, independently from the implementation
- Non-functional: measurable/perceivable properties of the systems not directly related to functional aspects

Requirements specification

- Textual description of system behaviour
- Basic specification technique
- Most used in practice
- ISO/IEC/IEEE standard 29148:2011 (E)
- Should be accessible from the IEEE digital library (https://ieeexplore.ieee.org/)
- (slides partly based on 5.2 “Requirements fundamentals”)
- ¨ ... I shall encourage you to read it !

Requirements Specification

→ How do we communicate the Requirements to others ?

- It is common practice to capture them in a specification
  - But an specification does not need to be a single paper document...

→ Purpose

- **Communication**
  - explains the application domain and the system to be developed
- **Contractual**
  - May be legally binding!
  - Expresses agreement and a commitment
- **Baseline for evaluating the software**
  - supports testing, V&V
  - "enough information to verify whether delivered system meets requirements"
- **Baseline for change control**

→ Audience

- **Customers & Users**
  - interested in system requirements…
  - …but not detailed software requirements
- **Systems (Requirements) Analysts**
  - Write other specifications that inter-relate
- **Developers, Programmers**
  - Have to implement the requirements
- **Testers**
  - Have to check that the requirements have been met
- **Project Managers**
  - Have to measure and control the project

Goal of a set of requirements

- enables an agreed understanding between stakeholders acquirers, users, customers, operators, suppliers
- is validated against real-world needs, can be implemented
- provides a basis of verifying designs and accepting solutions
- start with stakeholders intentions
- needs, goals, or objectives
- iterative process from stakeholders to system requirements

Organizing the Requirements

- Example Structures - organize by…
  - …**External stimulus or external situation**
    - e.g., for an aircraft landing system, each different type of landing situation: wind gusts, no fuel, short runway, etc
  - …**System feature**
    - e.g., for a telephone system: call forwarding, call blocking, conference call, etc
  - …**System response**
    - e.g., for a payroll system: generate pay-cheques, report costs, print tax info;
  - …**External object**
    - e.g. for a library information system, organize by book type
  - …**User type**
    - e.g. for a project support system: manager, technical staff, administrator, etc.
  - …**Mode**
    - e.g. for word processor: page layout mode, outline mode, text editing mode, etc
  - …**Subsystem**
    - e.g. for spacecraft: command&control, data handling, comms, instruments, etc.

Typical mistakes

- **Noise**
  - text that carries no relevant information to any feature of the problem.
- **Silence**
  - a feature that is not covered by any text.
- **Over-specification**
  - text that describes a detailed design decision, rather than the problem.
- **Contradiction**
  - text that defines a single feature in a number of incompatible ways.
- **Ambiguity**
  - text that can be interpreted in at least two different ways.
- **Forward reference**
  - text that refers to a terms or features yet to be defined.
- **Wishful thinking**
  - text that defines a feature that cannot possibly be validated.
- **Requirements on users**
  - Cannot require users to do certain things, can only assume that they will
- **Jigsaw puzzles**
  - distributing key information across a document and then cross-referencing
- **Duckspeak requirements**
  - Requirements that are only there to conform to standards
- **Unnecessary invention of terminology**
  - E.g. 'user input presentation function'
- **Inconsistent terminology**
  - Inventing and then changing terminology
- **Putting the onus on the developers**
  - i.e. making the reader work hard to decipher the intent
- **Writing for the hostile reader**
  - There are fewer of these than friendly readers

Do not write like this

→ **Ambiguity – *or***

↳ *The same subsystem shall also be able to generate visible* **or** *audible caution* **or** *warning signal for the attention of security* **or** *business analyst*

→ **Multiple requirements – *and, or, with, also***

↳ *The warning lamp shall light up when system intrusions is detected* **and** *the current workspace* **or** *input shall be saved*

Do not write like this

- **Let-out clauses**
  if, when, except, unless, although, always
  - *The fire alarm shall always be sounded* **when** *the smoke is detected,* **unless** *the alarm is being tested* **when** *the antivirus is deployed*

- **Long rumpling sentences**
  - *Provided that the designated input signals from the specified devices are received in the correct order where the systems is able to differentiate the designators, the security solution should comply with the required framework of Section 3.1.5 to indicate the desired security states*

Do not write like this

- **Speculation**
  usually, generally, often normally, typically
  - *Users* **normally** *require* **early** *indication of intrusion into the system*

- **Vague, undefinable terms**
  user-friendly, versatile, approximately, as possible, efficient, improved, high-performance, modern
  - *Security-related messages should be* **versatile** *and* **user-friendly**
  - *The OK status indicator lamp shall be illuminated* **as soon as possible** *after system security self-check is completed*

Do not write like this

→ **Wishful thinking**

100% reliable/ safe/ secure. Handle all unexpected failures. Please all users. Run on all platforms. Never fail. Upgrade to all future situations.

- *The gearbox shall be **100%** secure in normal operation.*
- *The network shall handle **all** unexpected errors without crashing.*

Good requirements

→ **Use simple direct sentences**

↳ *Security analyst should be able to view system status*

→ **Use a limited vocabulary**

↳ *Security analyst should be able* **to change** *the infected component in less than 12 hours*

↳ *Security analyst should be able* **to reconfigure** *the infected component in less than 12 hours*

Good requirements

- **Identify the type of user who wants each requirements**
  - The navigator shall be able to…
- **Focus on stating result**
  - … view storm clouds by radar …
- **Define verifiable criteria**
  - … at least 100 km ahead.
  - Acceptance criterion: Aircraft flying at 800km/h at 10.000 meters towards a known storm cloud indicated by meteorology satellite report; storm cloud is detected at a range of at least 100 km.

Criteria for Writing Good Requirements

- **What, not how (external observability)**
  - Avoid premature design or implementation decisions
- **Understandability, clarity (not ambiguous)**
- **Cohesiveness (one thing per requirement)**
- **Testability**
  - Somehow possible to test or validate whether the requirement has been met, clear acceptance criteria
  - Often requires quantification, this is more difficult for security than e.g. for performance
    - "The response time of function F should be max 2 seconds"
    - "The security of function F should be at least 99.9 %" ???

What is a requirement, actually

- is a statement expressing a need and its associated constraints and conditions
- is written in natural language
  - **structural language, "semi-formal"**
- it comprises a subject, a verb, a complement
  - **subject of the requirement**
  - **what shall be done**

Some syntax example (1)

- [Condition][Subject][Action][Object][Constraint]

- When signal x is received [**Condition**], the system [**Subject**] shall set [**Action**] the signal x received bit [**Object**] within 2 seconds [**Constraint**].

Some syntax example (2)

- [Condition][Subject][Action][Object][Constraint]

- At sea state 1 [**Condition**], the Radar System [**Subject**] shall detect [**Action**] targets [**Object**] at ranges out to 100 nautical miles [**Constraint**].

Some syntax example (3)

- [Subject][Action][Object][Constraint]

- The invoice system [**Subject**] shall display [**Action**] pending customer invoices [**Object**] in ascending order in which invoices are to be paid [**Constraint**].

Important points (1)

- Requirements:
  - **mandatory binding provisions and use ‘shall’ “deve”**
- Preferences and goals
  - **desired, non-mandatory, or non-binding use ‘should’ “dovrebbe”**
- Suggestions or allowance
  - **non-mandatory, non-binding, use ‘may’ “può”**
- Non-requirements, such as descriptive text
  - **use verbs such as ‘are’, ‘is’, ‘was’**
- avoid ‘must’ to prevent confusion with a requirement

Important points (2)

- Use positive statements
  - **avoid negative statement as ‘shall not’**
- Use active voice
  - **avoid passive voice such as ‘shall be able to detect’**
  - **write ‘shall detect’**
- In general, all terms specific to requirements should be clearly defined and applied consistently throughout all requirements of the system

Examples of constraints

- interfaces to already existing systems, where the interfaces cannot be change
  - e.g. format, protocol, or content
- physical size limitations
  - e.g. a controller shall fit within a limited space in an airplane wing
- laws of particular country
- pre-existing technology platform
- user or operator capabilities and limitations
- …

Single requirements characteristics (1)

- Necessary
  - requirement defines **essential capability**
  - if removed creates deficiency not fulfilled by other capabilities
  - requirement is applicable now, it is not obsolete
- Implementation free
  - **avoid unnecessary constraints** on the architectural design
  - requirement is about what - how is still open
- Unambiguous
  - only **one interpretation** - easy to understand
- Consistent
  - free of conflicts with other requirements

Non-conflicting

- R1: When the water level exceeds V, the system shall shut-down the water pipe.
- R2: When the fire sensor is activated, the system shall turn-on all water pipes.
- What happen if my house has R1 and R2 and a fire is detected?

Single requirements characteristics (2)

- Complete
  - no further amplification - sufficiently describes needs
  - measurable
- Singular
  - only one requirement - no conjunctions
- Feasible
  - technically achievable - no major technology advances needed
  - fits within system constraints
- Traceable
  - upwards and downwards
- Verifiable

Traceability matrix

» Means of expressing traceability information

Requirement	Design Elem.	Func	Test Case
SR-28	Class Catalog	sort	7, 8
SR-44	Class Catalog	import	12, 13

Two popular techniques

What are their advantages and disadvantages?

Requirement	Design element		
	Class Catalog	Class User	Class Book
SR-28	*		
SR-44	*		
SR-62		*	*
SR-73			*

Unbounded or ambiguous terms (1) (to be avoided!)

- Superlatives ('best', 'most', …)
- Subjective language ('user friendly', 'easy to use', 'cost effective', …)
- Vague pronouns ('it', 'this', 'that', …)
  - When module A calls B **its** history memory file is updated
- Ambiguous adverbs and adjectives ('significant', 'minimal', …)
- Open-ended, non-verifiable terms ('provide support', 'as a minimum', 'but not limited to', …)

Unbounded or ambiguous terms (2) (to be avoided!)

- Comparative phrases ('better than, 'higher quality', …)
- Loopholes ('if possible', 'as appropriate', 'as applicable', …)
- Incomplete references
- Negative statements (statement of capability not to be provided)